Theoretical Computer Science 410 (2009) 4026–4043

Contents lists available at ScienceDirect

Theoretical Computer Science

journal homepage: www.elsevier.com/locate/tcs

On determinism in modal transition systems

N. Beneš [1], J. Křetínský [1], K.G. Larsen, J. Srba *

Department of Computer Science, Aalborg University, Selma Lagerlöfs Vej 300, 9220 Aalborg Øst, Denmark

ARTICLE INFO

Keywords:
Compositional verification
Modal transition systems
Deterministic specifications
Refinement
Consistency

ABSTRACT

Modal transition system (MTS) is a formalism which extends the classical notion of labelled transition systems by introducing transitions of two types: must transitions that have to be present in any implementation of the MTS and may transitions that are allowed but not required.

The MTS framework has proved to be useful as a specification formalism of component-based systems as it supports compositional verification and stepwise refinement. Nevertheless, there are some limitations of the theory, namely that the naturally defined notions of modal refinement and modal composition are incomplete with respect to the semantic view based on the sets of the implementations of a given MTS specification. Recent work indicates that some of these limitations might be overcome by considering deterministic systems, which seem to be more manageable but still interesting for several application areas.

In the present article, we provide a comprehensive account of the MTS framework in the deterministic setting. We study a number of problems previously considered on MTS and point out to what extend we can expect better results under the restriction of determinism.

1. Introduction

The development of correct concurrent systems and processes constitutes a difficult and surprisingly subtle problem in computer science, having given rise to a number of proposed specification formalisms and verification methods over the years. The proposals may roughly be seen to fall within two main categories: the *logical approach*, in which a specification is a formula of some (temporal or modal) logic, and verification is a "model-checking" activity based on a denotational understanding of the specification; the *behavioural approach*, where specifications are objects of the same kind as implementations, in particular, specifications have operational interpretations. In this approach, verification is based on a comparison between the operational behaviours of the specification and implementation.

Ideally, we want a specification formalism that supports *stepwise refinement* and *component-based* development of systems. That is, starting from an initial specification, a series of small and successive refinements are made until eventually a specification is reached from which an implementation can be extracted directly. Each refinement step is relatively small, consisting typically in either conjoining additional requirements or in the replacement of a single component of the current specification with a more concrete/implementable one. In the latter case, the correctness of such a refinement step ought to be immediately implied by the correctness of the refinement of the replaced component, as this obviously will greatly simplify the task of verification. That is, we want our methodology to support *compositional* verification.

Also, we aim at *generality* in design and proofs: when designing a system there are often certain components or behavioural aspects which are beyond the scope (or control) of the design process—in particular third party components,

* Corresponding author. Tel.: +45 99 40 98 51; fax: +45 99 40 97 98.
E-mail addresses: xbenes3@fi.muni.cz (N. Beneš), xkretins@fi.muni.cz (J. Křetínský), kgl@cs.aau.dk (K.G. Larsen), srba@cs.aau.dk (J. Srba).
[1] Permanent address: Faculty of Informatics, Masaryk University, Botanická 68a, 60200 Brno, Czech Republic.

0304-3975/$ – see front matter © 2009 Elsevier B.V. All rights reserved.
doi:10.1016/j.tcs.2009.06.009

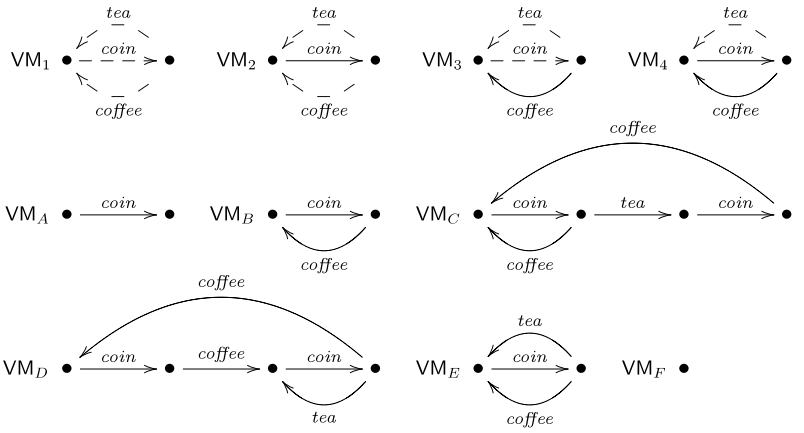

Fig. 1. Four specifications of a Vending Machine, VM_1–VM_4, and six different implementations VM_A–VM_F. Admissible transitions are shown using dashed arrows and required transitions are shown using full arrows.

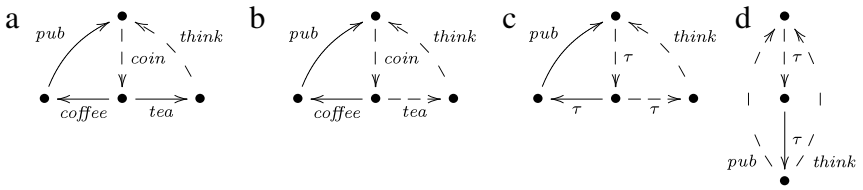

Fig. 2. Specification of a User (a), composition with VM_3 (b, c), and Determinization (d).

say. Thus it is necessary that the design and correctness proof of the system only rely on the (partial) specifications of these "uncontrollable" components.

Modal transition systems (MTS) were introduced some 20 years ago [1,2] by Larsen and Thomsen specifically in order to obtain an operational, yet expressive and manageable specification formalism meeting the above properties. In particular, MTSs are a variation of the classical model of labelled transition systems, where transitions come in two flavours: those that any refinement of the given specification *must* possess, and those that it *may, but is not required* to, have. As such, MTSs allow *loose* or *partial* specifications to be expressed, and enable the introduction of a *modal refinement* relation extending in a natural manner the classical notion of bisimulation on labelled transition systems. By *implementations* we then understand classical labelled transition systems (where may and must transitions coincide) that modally refine a given modal specification.

Viewing classical labelled transition systems as implementations, the four MTSs in Fig. 1 offer a series of vending machine specifications. VM_1 is very loose requiring nothing. VM_2 may be viewed as the preferred specification of the owner requiring implementations to have a coin-transition but it does not guarantee that there will afterwards be a coffee or a tea-transition. Similarly the coffee drinking customer's specification, VM_3, is a refinement of specification VM_1, requiring coffee after coin-insertion. VM_4 is a compromise refining both the owner and customer specifications—in fact it is the conjunction of the two specifications. Finally, VM_B–VM_E provide four, quite different, implementations of VM_4, varying in the degree of ability to offer tea to the user. Note that VM_A and VM_F do not implement VM_4, but VM_A implements VM_1 and VM_2, and VM_F implements VM_1 and VM_3. The notions of a modal refinement and of an implementation are formally introduced in Definitions 2.1 and 2.6.

Constructs for combining implementations (i.e. labelled transition systems) may be extended to MTSs in a straightforward manner. For example, Fig. 2(b, c) give a composition of a User with the vending machine VM_3, where synchronizations are either left unchanged or made invisible (using τ-actions). Fig. 2(a) specifies the type of User's who for sure will make a publication after having been given a cup of coffee. Given a cup of tea, on the other hand, the User needs additional time to think.

Semantically, we may identify an MTS specification with its set of implementations (i.e. the set of labelled transition systems refining it). The notions of modal refinement and modal composition are sound with respect to this semantic view. Thus whenever S is a modal refinement of T, then any implementation of S is indeed an implementation of T. Similarly, whenever P and Q are implementations of S and T (respectively) and $\oplus$ is a composition operator, then $P \oplus Q$ is an implementation of $S \oplus T$. On several occasions, these properties have proved sufficient in the stepwise and compositional development of concurrent systems guaranteed to be correct with respect to some given overall requirements.

However, as has been shown already in [1,3], both modal refinement and modal composition are *incomplete* with respect to the semantic view. In particular, there are MTSs S and T, where the set of implementations of S is included in that of T

without S being a modal refinement of T. Similarly, there are MTSs S and T, where the composed MTS $S \oplus T$ contains strictly more implementations than what can be obtained by composing implementations of S and T.

Recent results [4–7] characterizing the (high) complexity of semantic refinement (and semantic consistency) for MTSs point to the clear advantages of using the cheap notion of modal refinement (and modal composition) despite its incompleteness. Moreover, in most practical cases, where component specifications are *deterministic* – e.g. in our Vending Machine example and as advocated in the recent work by Henzinger and Sifakis [8,9] – modal refinement and modal composition seem to be complete, though they have not been studied in depth yet. In [8] the authors discuss two main challenges in embedded systems design: the challenge to build predictable systems, and that to build robust systems. They suggest how predictability can be formalized as a form of determinism, and robustness as a form of continuity. Thus, the purpose of this article is to make a thorough investigation of the MTS framework in the setting of determinism.

In particular, we study the completeness of modal refinement and modal composition for deterministic MTSs as well as some other questions related to the common implementation problem. As seen from our Vending Machine example (Fig. 2), the result of composing deterministic MTSs may well be a nondeterministic MTS. To allow the development and analysis to be continued using only deterministic MTSs, we provide a *determinization* construction on MTS, yielding for any given (possibly nondeterministic) MTS its least deterministic over-approximation.

The outline of the paper is as follows. In Section 2 we provide basic definitions of MTS as well as modal and semantic (thorough) refinements. Section 3 relates these notions of refinements with particular emphasis on deterministic MTSs. Section 4 shows the low complexity of both refinements in the deterministic case. Section 5 provides the complexity results for consistency (common implementation) between deterministic MTSs showing that consistency of a fixed number of specifications is NL-complete, whereas the complexity of consistency between an arbitrary number of MTSs remains hard even in the case of determinism (PSPACE-complete). Section 6 reconsiders the consistency problem in terms of the existence of a common deterministic implementation, showing that it is EXPTIME-complete. Finally, Section 7 considers the extension of composition operators to MTSs and shows the general lack of completeness even for deterministic MTSs; nevertheless, specific conditions guaranteeing completeness are identified.

2. Definitions

A *modal transition system* (MTS) over an action alphabet Σ is a triple $(P, \dashrightarrow, \longrightarrow)$, where P is a set of *processes* and $\longrightarrow \subseteq \dashrightarrow \subseteq P \times \Sigma \times P$ are *must* and *may* transition relations, respectively. The class of all MTSs is denoted by $\mathcal{MTS}$. We write $S \xrightarrow{a}$ if there exists some S' such that $S \xrightarrow{a} S'$, and $S \not\xrightarrow{a}$ if no such S' exists; similarly for $\dashrightarrow$.

An MTS is *deterministic* if for each $S \in P$ and $a \in \Sigma$ there is at most one S' such that $S \overset{a}{\dashrightarrow} S'$. The class of all deterministic MTSs is denoted $d\mathcal{MTS}$.

An MTS is an *implementation* if $\dashrightarrow \;=\; \longrightarrow$. The class of all implementations is denoted by $i\mathcal{MTS}$. Note that because in implementations the must and may relations coincide, we can consider such systems as the standard *labelled transition systems*.

We use capital letters for processes and calligraphic letters for sets of processes. Moreover, letters $S, T, U, \ldots$ are used to denote processes in general, letters $D, E, F, \ldots$ are reserved for deterministic processes, and letters $I, J, \ldots$ are used to denote implementations.

Because in MTS whenever $S \xrightarrow{a} S'$ then necessarily also $S \overset{a}{\dashrightarrow} S'$, we adopt the convention of not drawing may transitions between processes where must transitions are present.

Whenever clear from the context, we refer to processes without explicitly mentioning their underlying MTSs. We also write e.g. $S \in d\mathcal{MTS}$, meaning that the underlying MTS of the process S is in $d\mathcal{MTS}$.

Definition 2.1. Let $M_1 = (P_1, \dashrightarrow_1, \longrightarrow_1)$, $M_2 = (P_2, \dashrightarrow_2, \longrightarrow_2)$ be MTSs over the same action alphabet and $S \in P_1, T \in P_2$ be processes. We say that S *modally refines* T, written $S \leq_m T$, if there is a relation $R \subseteq P_1 \times P_2$ such that $(S, T) \in R$ and for every $(A, B) \in R$ and every $a \in \Sigma$:

1. if $A \overset{a}{\dashrightarrow}_1 A'$ then there is a transition $B \overset{a}{\dashrightarrow}_2 B'$ s.t. $(A', B') \in R$, and
2. if $B \xrightarrow{a}_2 B'$ then there is a transition $A \xrightarrow{a}_1 A'$ s.t. $(A', B') \in R$.

We often omit the indices in the transition relations and use symbols $\dashrightarrow$ and $\longrightarrow$ whenever it is clear from the context what transition system we have in mind.

Remark 2.2. Note that on implementations modal refinement coincides with the classical notion of strong bisimilarity, and on modal transition systems without any must transitions it corresponds to the well-studied simulation preorder.

We will now extend the standard game-theoretic characterization of bisimilarity [10,11] to the game characterization of modal refinement.

A *modal refinement game* (or simply a *modal game*) on a pair of processes (S, T) is a two-player game between *Attacker* and *Defender*. The game is played in *rounds*. In each round the players change the *current pair of processes* (A, B) (initially $A = S$ and $B = T$) according to the following rule:

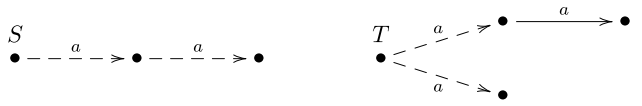

Fig. 3. $S \leq_t T$, but $S \not\leq_m T$.

1. Attacker chooses an action $a \in \Sigma$ and one of the processes A or B. If he chose A then he performs a move $A \xrightarrow{a} A'$ for some A'; if he chose B then he performs a move $B \xrightarrow{a} B'$ for some B'.
2. Defender responds by choosing a transition under a in the other process. If Attacker chose the move from A, Defender has to answer by a move $B \xrightarrow{a} B'$ for some B'; if Attacker chose the move from B, Defender has to answer by a move $A \xrightarrow{a} A'$ for some A'.
3. The new current pair of processes becomes (A', B') and the game continues with a next round.

The game is similar to standard bisimulation game with the exception that Attacker is only allowed to attack on the left-hand side using may transitions (and Defender answers by may transitions on the other side), while on the right-hand side Attacker attacks using must transitions (and Defender answers by must transitions in the left-hand side process).

Any *play* (of the modal game) thus corresponds to a sequence of pairs of processes formed according to the above rule. A play (and the corresponding sequence) is finite iff one of the players gets stuck (cannot make a move). The player who got stuck lost the play and the other player is the winner. If the play is infinite then Defender is the winner.

The following proposition is by a standard argument in analogy with strong bisimulation games (see also [10,11]).

Proposition 2.3. *It holds that $S \leq_m T$ iff Defender has a winning strategy in the modal game starting with the pair (S, T); and $S \not\leq_m T$ iff Attacker has a winning strategy.*

Example 2.4. Consider processes S and T in Fig. 3. We prove that S does not modally refine T. Indeed, Attacker has the following winning strategy in the modal game starting from (S, T). Attacker plays the may transition under the action a on the left-hand side process S and Defender can answer by entering either the upper or lower branch in the process T. In the first case Attacker wins by playing the must transition under a on the right-hand side, for which Defender has no answer on the left-hand side (no must transition under a is available) and loses. In the second case Attacker wins by playing the second may transition under a in the left-hand side process and Defender loses as well.

We shall now observe that the modal refinement problem, i.e. the question whether a given process modally refines another given process, is tractable for finite MTSs.

Theorem 2.5. *The modal refinement problem for finite MTSs is P-complete.*

Proof. Modal refinement can be computed in P by the standard greatest fixed-point computation, similarly as in the case of strong bisimulation (for efficient algorithms implementing this strategy see e.g. [12,13]). P-hardness of modal refinement follows from the P-hardness of bisimulation ([14], see also [15]). □

We proceed with the definition of thorough refinement, a relation that holds for two modal specification S and T iff any implementation of S is also an implementation of T. This relation is of our major interest since it captures the semantic point of view.

Definition 2.6. For a process S let us denote by $[\![S]\!] = \{I \in i\mathcal{MTS} \mid I \leq_m S\}$ the set of all implementations of S. We say that *S thoroughly refines T*, written $S \leq_t T$, if $[\![S]\!] \subseteq [\![T]\!]$.

The following two observations are trivial.

Lemma 2.7. *Relations $\leq_m$ and $\leq_t$ are transitive.*

Lemma 2.8. *Let $I, J \in i\mathcal{MTS}$. Then $I \leq_m J$ if and only if $I \leq_t J$; and both $\leq_m$ and $\leq_t$ coincide with strong bisimilarity.*

3. Modal and thorough refinements

In this section we investigate several properties of modal and thorough refinements, with a particular focus on deterministic processes. First, we observe that thorough refinement is implied by the modal refinement, irrelevant whether the processes are deterministic or not.

Lemma 3.1. *Let S, T be processes. If $S \leq_m T$ then $S \leq_t T$.*

Proof. For $I \in [\![S]\!]$ we have $I \leq_m S \leq_m T$, hence $I \leq_m T$ by Lemma 2.7 and thus $I \in [\![T]\!]$. □

Remark 3.2. The opposite direction in Lemma 3.1 does not hold as we demonstrate in Fig. 3. In Example 2.4 we already argued that $S \not\leq_m T$. However, S has only implementations that can perform at most two consecutive a-actions. As any such implementation is clearly also an implementation of T, we conclude that $S \leq_t T$.

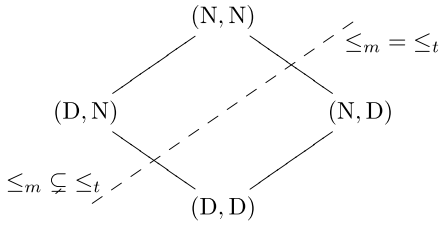

Fig. 4. Relationship between refinements on determin. (D) and nondetermin. (N) systems.

The fact that thorough refinement does not imply modal refinement might be considered as a limitation of the theory developed in the previous studies on modal transition systems. Nevertheless, in the context of deterministic systems, we show that thorough and modal refinement coincide, provided that the right-hand side process is deterministic.

Lemma 3.3. *Let S, D be processes and $D \in d\mathcal{MTS}$. If $S \leq_t D$ then $S \leq_m D$.*

Proof. Assume that $S \leq_t D$ and that D is deterministic. We define a relation R that satisfies the conditions of Definition 2.1. The relation R is taken as the smallest relation such that $(S, D) \in R$ and whenever $(T, E) \in R$, $T \xrightarrow{a} T'$ and $E \xrightarrow{a} E'$ for some a then also $(T', E') \in R$. The relation R is clearly well defined. Before we prove that R satisfies the refinement conditions, we make the claim that $(T, E) \in R$ implies $T \leq_t E$. Clearly, this holds for (S, D). Suppose now that $T \leq_t E$, $T \xrightarrow{a} T'$, $E \xrightarrow{a} E'$ and I' is an arbitrary implementation of T'. Then there exists some implementation $I \in [\![T]\!]$ such that $I \xrightarrow{a} I'$. But as $T \leq_t E$, I is also an implementation of E. Therefore, as E is deterministic, I' is an implementation of E', thus $T' \leq_t E'$. We can now check that R is a modal refinement relation. Let $(T, E) \in R$.

(i) Suppose that $T \xrightarrow{a} T'$. Then, there exists an implementation $I \in [\![T]\!]$ that has an $\xrightarrow{a}$ transition. As $T \leq_t E$, I is also an implementation of E and therefore $E \xrightarrow{a} E'$ for some E'. By the definition of R, $(T', E') \in R$.

(ii) Suppose that $E \xrightarrow{a} E'$. Then, all implementations of E are forced to have an $\xrightarrow{a}$ transition. As $T \leq_t E$, this implies that all implementations of T have an $\xrightarrow{a}$ transition. Therefore, $T \xrightarrow{a} T'$ for some T' and $(T', E') \in R$ by the definition of R. □

The claim of Lemma 3.3 does not hold for the inverse case where the refining process is deterministic and the refined process is arbitrary. The counterexample to this claim was already shown in Fig. 3. Fig. 4 summarizes the known relationships between thorough and modal refinement for all possible cases of (non)determinism of the two systems. The conclusion is that whenever the right-hand side process is deterministic, modal and thorough refinement relations coincide. If the right-hand side process can be nondeterministic, modal refinement is a strictly stronger relation than thorough refinement.

The modal refinement can be checked in polynomial time, as we know from Theorem 2.5, but the thorough refinement is PSPACE-hard in general [5] (it is moreover shown in [5] that this problem is in EXPTIME). Therefore, there is a clear motivation to approximate processes by deterministic ones, in order to be able to use faster modal refinement procedures instead (at least for the instances where the deterministic approximation of a process is not exponentially larger).

For any two (in general nondeterministic) processes S and T, we have that $S \leq_m T$ implies $S \leq_t T$. The converse is not true in general, but we will define a monotone deterministic over-approximation operator $\mathcal{D}$, so that $S \leq_t T$ implies $\mathcal{D}(S) \leq_m \mathcal{D}(T)$ (as stated formally later on in Lemma 3.6). Moreover, we show that there exists a *smallest* (w.r.t. refinement) deterministic system refined by the original system. We call it the *deterministic hull*.

Definition 3.4 (*Construction of the Deterministic Hull*). Let S be an arbitrary process with $(P, \dashrightarrow, \longrightarrow)$ being its underlying MTS. The *deterministic hull* of S, denoted by $\mathcal{D}(S)$, is constructed by a modal extension of the standard subset construction. For $\emptyset \neq \mathcal{T} \subseteq P$ and an action a let $\mathcal{T}_a = \{T' \in P \mid \exists T \in \mathcal{T} : T \xrightarrow{a} T'\}$ be the set of all may-successors under the action a. We define an MTS $M = (\mathcal{P}(P) \setminus \{\emptyset\}, \dashrightarrow_{\mathrm{D}}, \longrightarrow_{\mathrm{D}})$ where transitions are given as follows:

(i) if $\mathcal{T}_a \neq \emptyset$, we set $\mathcal{T} \xrightarrow{a}_{\mathrm{D}} \mathcal{T}_a$, and

(ii) if moreover for all $T \in \mathcal{T}$ there exists some $T' \in \mathcal{T}_a$ such that $T \xrightarrow{a} T'$, then we set also $\mathcal{T} \xrightarrow{a}_{\mathrm{D}} \mathcal{T}_a$.

There are no other transitions. Then, the process $\mathcal{D}(S)$ is defined as the singleton set containing S, i.e. $\mathcal{D}(S) = \{S\}$.

An example of this construction is given in Fig. 5.

Theorem 3.5 (*Soundness and Minimality of $\mathcal{D}(S)$ Construction*). *Let S be an arbitrary process. Then $\mathcal{D}(S)$ is a deterministic process such that $S \leq_t \mathcal{D}(S)$ and for every $D \in d\mathcal{MTS}$, if $S \leq_t D$ then $\mathcal{D}(S) \leq_t D$.*

Proof. The fact that $\mathcal{D}(S)$ is deterministic for any S is clear from the construction. The first claim we need to prove is that $S \leq_t \mathcal{D}(S)$. We will do so by showing that $S \leq_m \mathcal{D}(S)$ (note that by Lemma 3.1 this implies that $S \leq_t \mathcal{D}(S)$). We define the refinement relation R such that $(S, \mathcal{T}) \in R$ iff $S \in \mathcal{T}$ and we need to prove that it satisfies the conditions of Definition 2.1.

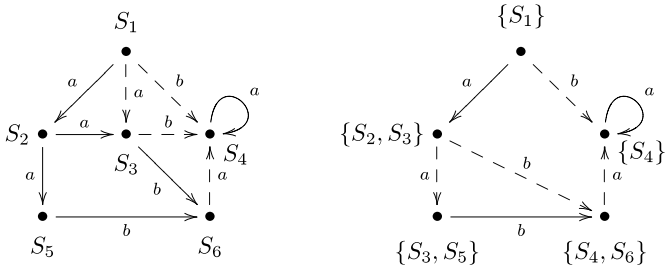

Fig. 5. A process and its deterministic hull $\mathcal{D}(S_1) = \{S_1\}$.

Clearly $(S, \mathcal{D}(S)) \in R$. Now let $(S, \mathcal{T}) \in R$. On the one hand, suppose that $S \overset{a}{\dashrightarrow} S'$. Then clearly from the previous construction $\mathcal{T} \overset{a}{\dashrightarrow}_{\mathrm{D}} \mathcal{T}_a$ and $S' \in \mathcal{T}_a$, thus $(S', \mathcal{T}_a) \in R$. On the other hand, suppose that $\mathcal{T} \overset{a}{\longrightarrow}_{\mathrm{D}} \mathcal{T}'$. It follows from the construction that $\mathcal{T}' = \mathcal{T}_a$, $S \overset{a}{\longrightarrow} S'$ for some S' and that $S' \in \mathcal{T}_a$, thus $(S', \mathcal{T}_a) \in R$. Hence $S \leq_m \mathcal{D}(S)$.

Now, we need to prove the minimality of the deterministic hull, i.e. that for each deterministic D such that $S \leq_t D$ we also get $\mathcal{D}(S) \leq_t D$. As for deterministic processes on the right-hand side modal and thorough refinements coincide (Lemmas 3.1 and 3.3), it is enough to prove the minimality w.r.t. $\leq_m$.

Let D be a deterministic process such that $S \leq_m D$. This means that there is a relation R satisfying the conditions of Definition 2.1. We show that $\mathcal{D}(S) \leq_m D$ by constructing a new relation Q that also satisfies these conditions. The definition of Q is as follows:

$$(\mathcal{T}, E) \in Q \quad \text{if and only if} \quad \emptyset \neq \mathcal{T} \subseteq \{T \mid (T, E) \in R\}.$$

It remains to be proved that Q satisfies the refinement relation conditions. Since $(S, D) \in R$, we have $(\mathcal{D}(S), D) = (\{S\}, D) \in Q$. Now, let $(\mathcal{T}, E) \in Q$.

On the one hand, suppose that $\mathcal{T} \overset{a}{\dashrightarrow}_{\mathrm{D}} \mathcal{T}'$. Then for each $T' \in \mathcal{T}'$, there is at least one $T \in \mathcal{T}$ such that $T \overset{a}{\dashrightarrow} T'$ (as $\mathcal{T}' = \mathcal{T}_a$). Because $(T, E) \in R$, there is E' such that $E \overset{a}{\dashrightarrow} E'$ with $(T', E') \in R$. Moreover, as E is deterministic, this E' is unique and the same for all $T' \in \mathcal{T}'$, thus $(\mathcal{T}', E') \in Q$.

On the other hand, suppose that $E \overset{a}{\longrightarrow} E'$. Then, for all T such that $(T, E) \in R$, there has to be some T' such that $T \overset{a}{\longrightarrow} T'$ with $(T', E') \in R$. Moreover, as E is deterministic, it holds that for all T with $(T, E) \in R$, whenever $T \overset{a}{\dashrightarrow} T'$ then $(T', E') \in R$. This implies that $\mathcal{T} \overset{a}{\longrightarrow}_{\mathrm{D}} \mathcal{T}_a$, as for each $T \in \mathcal{T}$ there is an outgoing $\overset{a}{\longrightarrow}$ transition, and clearly $\mathcal{T}_a \subseteq \{T' \mid (T', E') \in R\}$, thus $(\mathcal{T}_a, E') \in Q$. Therefore, $\mathcal{D}(S) \leq_m D$. □

Lemma 3.6. *Let S, T be processes. If $S \leq_t T$ then $\mathcal{D}(S) \leq_m \mathcal{D}(T)$.*

Proof. Let $S \leq_t T$. By Theorem 3.5 we know that $T \leq_t \mathcal{D}(T)$ and from the transitivity of $\leq_t$ also $S \leq_t \mathcal{D}(T)$. By the minimality of $\mathcal{D}(S)$ (Theorem 3.5) we get $\mathcal{D}(S) \leq_t \mathcal{D}(T)$ and by Lemma 3.3 we conclude with $\mathcal{D}(S) \leq_m \mathcal{D}(T)$. □

Finally, note that the construction of the deterministic hull on MTSs which contain only may transitions is the same as the determinization of finite automata. Therefore, the example of an exponential blow-up in the size [16, page 65] carries over to our setting and thus the deterministic hull $\mathcal{D}(S)$ might be of exponential size w.r.t. to some particular finite nondeterministic process S.

4. Complexity results for refinement problems

In this section we study the following decision problems of modal and thorough refinement and argue about their complexity. Recall that we use the notation where D, E stand for deterministic processes and S, T for general processes. Moreover, throughout Sections 4–6 which deal with complexity, all processes are implicitly assumed to be defined over finite MTS.

$$\begin{aligned}
\mathrm{MR}_{\mathrm{D,D}} &= \{\langle D, E\rangle \mid D \leq_m E\} & \mathrm{TR}_{\mathrm{D,D}} &= \{\langle D, E\rangle \mid D \leq_t E\}\\
\mathrm{MR}_{\mathrm{D,N}} &= \{\langle D, S\rangle \mid D \leq_m S\} & \mathrm{TR}_{\mathrm{D,N}} &= \{\langle D, S\rangle \mid D \leq_t S\}\\
\mathrm{MR}_{\mathrm{N,D}} &= \{\langle S, D\rangle \mid S \leq_m D\} & \mathrm{TR}_{\mathrm{N,D}} &= \{\langle S, D\rangle \mid S \leq_t D\}\\
\mathrm{MR}_{\mathrm{N,N}} &= \{\langle S, T\rangle \mid S \leq_m T\} & \mathrm{TR}_{\mathrm{N,N}} &= \{\langle S, T\rangle \mid S \leq_t T\}.
\end{aligned}$$

By Lemmas 3.1 and 3.3 we know that $\mathrm{MR}_{\mathrm{D,D}} = \mathrm{TR}_{\mathrm{D,D}}$ and $\mathrm{MR}_{\mathrm{N,D}} = \mathrm{TR}_{\mathrm{N,D}}$.

Our first result in this section says that modal refinement is decidable in nondeterministic logarithmic space, provided that the right-hand side process is deterministic.

Theorem 4.1. *The problem $\mathrm{MR}_{\mathrm{N,D}}$ is in NL.*

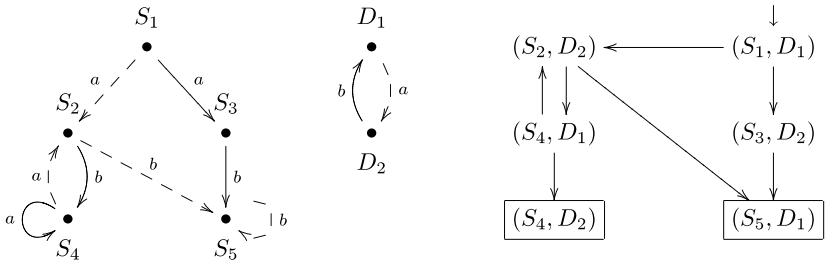

Fig. 6. An example of two MTSs and the corresponding graph (marked nodes are in a box).

In order to prove the above theorem, let S be an arbitrary process and let D be a deterministic one. We will show that the problem of deciding $S \leq_m D$ is in NL by reduction to the graph reachability problem, known to be NL-complete [17]. Note that we are actually reducing the problem whether $S \not\leq_m D$ to the graph reachability problem. However, this poses no problem, as the NL complexity class is closed under complement.

The graph will be constructed in the following way. The nodes of the graph will be all pairs (T, E) where T is a process of the MTS for S and E is a process of the MTS for D. There are three kinds of nodes.

(i) Nodes (T, E) such that $T \overset{a}{\dashrightarrow}$ and $E \overset{a}{\dashrightarrow\!\!\!\!\!/}$ for some action a. Such nodes have no outgoing edges and are called *marked*.
(ii) Nodes (T, E) such that $E \overset{a}{\longrightarrow}$ and $T \overset{a}{\longrightarrow\!\!\!\!\!/}$ for some action a. As in the previous case, such nodes have no outgoing edges and are called *marked*.
(iii) Nodes (T, E) which do not satisfy conditions (i) or (ii). Such nodes are called *unmarked* and there is an edge from (T, E) to (T', E') whenever $T \overset{a}{\dashrightarrow} T'$ and $E \overset{a}{\dashrightarrow} E'$ for some action a.

An example illustrating the reduction is given in Fig. 6. We now prove the correctness of the reduction.

Lemma 4.2. *We have $S \not\leq_m D$ if and only if a marked node is reachable from the node (S, D).*

Proof. For the if case, suppose that there is a marked node reachable from (S, D), i.e. there exists a path $(S, D) = (T_0, E_0), (T_1, E_1), \ldots, (T_n, E_n)$ where (T_n, E_n) is marked. We can easily show that Attacker has a winning strategy in the modal game played from (S, D). Attacker will simply play in the left-hand side process S following the sequence $S = T_0, T_1, \ldots, T_n$ under the may transitions. Because the right-hand side process is deterministic, Defender can only answer by going through the processes $D = E_0, E_1, \ldots, E_n$. From the pair (T_n, E_n) Attacker now easily wins. If the pair was marked due to condition (i), then Attacker chooses an action a and an arbitrary transition $T_n \overset{a}{\dashrightarrow}$ to which Defender is unable to respond to. Likewise, if the pair was marked due to condition (ii) above, then Attacker chooses an action a on the right-hand side and the unique transition $E_n \overset{a}{\longrightarrow}$. Again, Defender has no response and loses.

For the only if case, suppose that no marked nodes are reachable from (S, D). We show a relation R that satisfies the conditions of Definition 2.1. The relation R is defined as

$$R = \{(T, E) \mid (T, E) \text{ is reachable from } (S, D) \text{ in the graph}\}.$$

Clearly, $(S, D) \in R$. Now suppose that $(T, E) \in R$. If $T \overset{a}{\dashrightarrow} T'$ then also, as (T, E) is unmarked, $E \overset{a}{\dashrightarrow} E'$ and moreover, $(T', E') \in R$ due to the definition of the graph. For the other condition, suppose that $E \overset{a}{\longrightarrow} E'$. Then, again because (T, E) is unmarked, also $T \overset{a}{\longrightarrow} T'$ for some T' and $(T', E') \in R$ from the definition of the graph. Thus, $S \leq_m D$. □

We have thus shown that the $\mathrm{MR}_{\mathrm{N,D}}$ problem is in NL. The next theorem establishes NL-hardness even for the $\mathrm{MR}_{\mathrm{D,D}}$ problem.

Theorem 4.3. *The problem $\mathrm{MR}_{\mathrm{D,D}}$ is NL-hard.*

Proof. The proof is done by reduction from the graph reachability problem to $\mathrm{MR}_{\mathrm{D,D}}$. In fact, there is a folklore result that strong bisimilarity on finite and deterministic processes is NL-hard, which immediately implies our theorem. Nevertheless, for the self-containment of the presentation, we sketch a simple construction demonstrating this fact.

Assume a given graph G with a source and a target node. The main idea is that we make two identical copies of the graph G and treat them like implementations I_1 and I_2. These implementations must be deterministic, but this can be easily accomplished by taking a fixed ordering of successors of each node in the original graph and by labelling the transitions in the implementations with natural numbers accordingly. The two deterministic implementations I_1 and I_2 now differ only in one detail. In I_1 we introduce a loop on the target node under some fresh action. Clearly, the target node is reachable in G iff $I_1 \not\leq_m I_2$. Thus $\mathrm{MR}_{\mathrm{D,D}}$ is NL-hard. □

Corollary 4.4. *Problems $\mathrm{MR}_{\mathrm{D,D}}$, $\mathrm{TR}_{\mathrm{D,D}}$, $\mathrm{MR}_{\mathrm{N,D}}$, $\mathrm{TR}_{\mathrm{N,D}}$ are NL-complete.*

Proof. By Lemmas 3.1, 3.3, Theorems 4.1 and 4.3. □

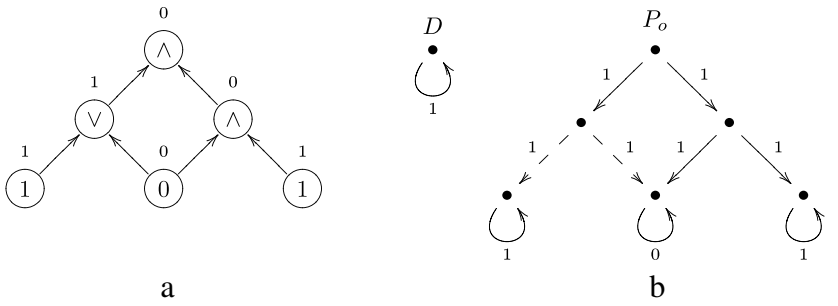

Fig. 7. (a) A negative instance of mCVP. (b) Processes D and P_o such that $D \not\leq_m P_o$.

We now continue with studying the complexity of modal refinement for the situation when the right-hand side process may be nondeterministic. First, we prove P-hardness of the problem $\mathrm{MR}_{\mathrm{D,N}}$. Note that this fact does not directly follow from P-hardness of strong bisimilarity because the reductions provided in [14,15] use nondeterministic systems on both sides.

Theorem 4.5. *The problem* $\mathrm{MR}_{\mathrm{D,N}}$ *is P-hard.*

The proof is done by reduction from a P-complete problem mCVP (monotone circuit value problem) [17]. A *monotone Boolean circuit* is a finite directed acyclic graph in which the nodes are either of indegree zero (*input* nodes) or of indegree two and there is exactly one node of outdegree zero (*output* node). Each non-input node is labelled either with $\wedge$ or $\vee$. An *input of the circuit* is an assignment of values 0 or 1 to the input nodes. Given an input, the circuit computes the output value as follows: the value of an input node is given by the input assignment, the value of a node labelled with $\wedge$ or $\vee$ is the conjunction or disjunction of values of its predecessors, respectively. The *output value of the circuit* is the value of the output node. The mCVP problem is, given a monotone Boolean circuit and its input, to decide whether the output value is 1. An example of a monotone Boolean circuit with an input assignment and computed values at each node is given in Fig. 7(a).

Given a monotone Boolean circuit and its input, we construct two processes D and P_o. The process D has only two transitions $D \xrightarrow{1} D$ and $D \overset{1}{\dashrightarrow} D$. The process P_o is constructed as follows. For each input node u we add a process P_u with the loops $P_u \xrightarrow{b} P_u$ and $P_u \overset{b}{\dashrightarrow} P_u$ where b is the value assigned to the node u. For each node v labelled with $\wedge$ we add a process P_v with transitions $P_v \xrightarrow{1} P_{v'}$, $P_v \overset{1}{\dashrightarrow} P_{v'}$, $P_v \xrightarrow{1} P_{v''}$ and $P_v \overset{1}{\dashrightarrow} P_{v''}$ where v' and v'' are the predecessors of v in the Boolean circuit. Similarly, for each node w labelled with $\vee$ we add a process P_w with transitions $P_w \overset{1}{\dashrightarrow} P_{w'}$ and $P_w \overset{1}{\dashrightarrow} P_{w''}$ where w' and w'' are the predecessors of w. We assume that P_o denotes the process representing the output node of the circuit.

The reduction for the mCVP of Fig. 7(a) is illustrated in Fig. 7(b). We now show the correctness of the reduction.

Lemma 4.6. *The output value of the circuit is 1 if and only if* $D \leq_m P_o$.

Proof. For the if case, suppose that the output value of the circuit is 0. We show that Attacker has a winning strategy in the modal game starting from (D, P_o). From each current pair (D, P_v) Attacker decides what to play according to the type of node v. If v is labelled with $\wedge$, then at least one predecessor of v has value 0, say w, and Attacker chooses $P_v \xrightarrow{1} P_w$, to which the Defender responds by playing $D \xrightarrow{1} D$. If v is labelled with $\vee$, then all predecessors of v have value 0. Attacker then chooses $D \overset{1}{\dashrightarrow} D$, to which the Defender responds with any of the two possibilities. Clearly, this way the play only proceeds through pairs of processes (D, P_v) where v has the value 0 and finally it arrives into the pair (D, P_i) where i is an input node with assigned value 0. Attacker then plays $P_i \xrightarrow{0} P_i$ to which Defender has no response and Attacker wins.

For the only if case, suppose that the output value of the circuit is 1. We define a modal refinement relation R by

$$R = \{(D, P_v) \mid v \text{ is a node with value } 1\}.$$

Clearly, $(D, P_o) \in R$ as the output of the circuit is 1. Now suppose that $(D, P_v) \in R$. The only may transition of D is $D \overset{1}{\dashrightarrow} D$. If v is an input node then it is an input with assigned value of 1 and then $P_v \overset{1}{\dashrightarrow} P_v$ and $(D, P_v) \in R$. If v is a non-input node then it has to have at least one predecessor with value 1 (otherwise it could not have the value of 1 itself), say u. Then $P_v \overset{1}{\dashrightarrow} P_u$ and $(D, P_u) \in R$. For the other part, suppose that $P_v \xrightarrow{1} P_w$. But $D \xrightarrow{1} D$ and the must transition of P_v implies that v is labelled with $\wedge$, therefore all its predecessors must have the value of 1 and $(D, P_w) \in R$. Thus $D \leq_m P_o$. □

After we have shown P-hardness of the $\mathrm{MR}_{\mathrm{D,N}}$ problem, we can conclude with the following corollary of Theorems 4.5 and 2.5.

Corollary 4.7. *The problems* $\mathrm{MR}_{\mathrm{D,N}}$ *and* $\mathrm{MR}_{\mathrm{N,N}}$ *are P-complete.*

Note that the complexity of the $\mathrm{TR}_{\mathrm{N,N}}$ problem was recently settled to EXPTIME-completeness [4], improving thus the previously known PSPACE-hardness result [5]. Regarding the $\mathrm{TR}_{\mathrm{D,N}}$ problem we know only its containment in EXPTIME and co-NP-hardness [7].

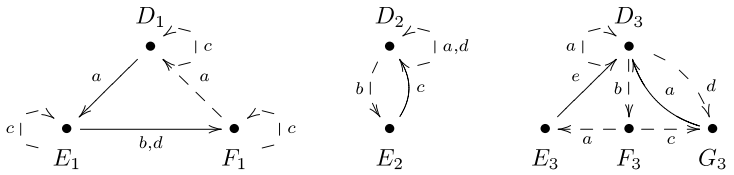

Fig. 8. Do (D_1, D_2, D_3) have a common implementation?.

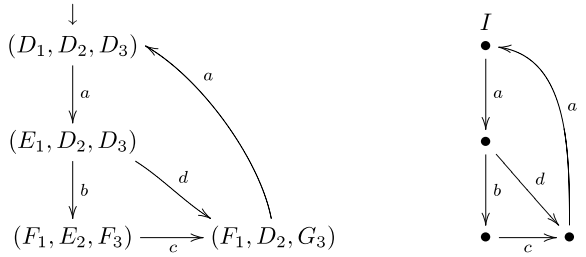

Fig. 9. The graph constructed for (D_1, D_2, D_3) and the corresponding common implementation I of (D_1, D_2, D_3).

5. Complexity results for common implementation problem

The common implementation problem (CI for short) is the problem of deciding, given two or more processes of modal transition systems, whether there is a single process that implements all these processes at the same time. For the general case, where processes can be nondeterministic, it is known that the CI problem is EXPTIME-complete [6] and if the number of the processes is fixed, the problem is P-complete. The containment in P is proved in [18] and P-hardness follows from [14,15], as bisimilarity is a special case of CI for two processes where both processes are implementations. We will now look at a specialized variant of this problem, where the given processes are assumed to be *deterministic*. This restricted problem is called $\mathrm{CI_D}$ (or $\mathrm{CI_D^k}$ if the number of processes is fixed to be k) and its formal definition is as follows.

$$\mathrm{CI_D^k} = \{\langle D_1, \ldots, D_k\rangle \mid \exists I : I \in [\![D_1]\!] \cap \cdots \cap [\![D_k]\!] \text{ and } D_1, \ldots, D_k \in d\mathcal{MTS}\}$$
$$\mathrm{CI_D} = \bigcup_{k=2}^{\infty} \mathrm{CI_D^k}.$$

When given an instance of the $\mathrm{CI_D}$ problem, we will use two parameters to describe its size: k the number of the processes and $n = |D_1| + |D_2| + \cdots + |D_k|$ the size of the whole input.

Our first complexity result is that the existence of a common implementation for processes $D_1, \ldots, D_k$ can be decided in nondeterministic $\mathcal{O}(k \log n)$ space. Proving this claim will give us the following theorem.

Theorem 5.1. *The problem* $\mathrm{CI_D}$ *is in PSPACE. The problem* $\mathrm{CI_D^k}$ *(for any fixed k) is in NL.*

We show this by reducing the problem of nonexistence of common implementation to the graph reachability problem, which is known to be decidable in nondeterministic $\mathcal{O}(\log N)$ space, where N is the size of the graph [17]. The graph we are going to construct is of size n^k and moreover, the construction can be done on the fly, so that no additional space is needed, thus yielding the result.

The graph we are going to construct will have labels on its edges. This is a technical detail that does not influence the complexity of the graph reachability problem, but will prove useful later, when we discuss the correctness. The basic idea of the construction is that the graph represents an implementation that in each step includes only those transitions that are required by at least one of the processes. The marked nodes then represent situations where all these requirements are impossible to satisfy.

The construction is done in the following way. Each node of the graph is a k-tuple $(E_1, \ldots, E_k)$, where E_i is a process of the MTS underlying D_i for all i. The initial node is $(D_1, \ldots, D_k)$. Nodes $(E_1, \ldots, E_k)$ where there exists an action a such that $E_i \xrightarrow{a} E_i'$ for some i, but some E_j has no outgoing $\xrightarrow{a}$ transition, are considered *marked*. The edges in the graph are defined as follows.

$$(E_1, \ldots, E_k) \xrightarrow{a} (F_1, \ldots, F_k) \iff \forall i : E_i \xrightarrow{a} F_i \quad \text{and} \quad \exists j : E_j \xrightarrow{a} F_j.$$

The construction is illustrated in Figs. 8 and 9, where Fig. 9 contains only the nodes reachable from (D_1, D_2, D_3).

We shall now prove the following lemma that asserts correctness of this reduction.

Lemma 5.2. *Processes* $D_1, \ldots, D_k$ *have a common implementation if and only if there are no marked nodes reachable from the node* $(D_1, \ldots, D_k)$.

Proof. For the if case, suppose that there are no marked nodes reachable from $(D_1, \ldots, D_k)$. We show a common implementation of all D_i. As it has labels on its edges, the graph itself may be seen as an MTS where $\longrightarrow = \dashrightarrow$. Then, the node $(D_1, \ldots, D_k)$ may be seen as a process. We show that this process is a common implementation of all D_i, $1 \leq i \leq k$.

Let us so fix any number i from 1 to k. We define

$$R_i = \left\{\left((E_1, \ldots, E_k), E_i\right) \mid (E_1, \ldots, E_k) \text{ is a node in the graph}\right\}$$

and show that R_i is a relation of modal refinement. Clearly $((D_1, \ldots, D_k), D_i) \in R_i$. Suppose that $((E_1, \ldots, E_k), E_i) \in R_i$. If it is the case that $(E_1, \ldots, E_k) \overset{a}{\dashrightarrow} (F_1, \ldots, F_k)$ then clearly $E_i \overset{a}{\dashrightarrow} F_i$ by the definition. Conversely, if $E_i \overset{a}{\longrightarrow} F_i$ then, as $(E_1, \ldots, E_k)$ is not marked, all E_j have an $\overset{a}{\dashrightarrow}$ transition to some F_j and therefore $(E_1, \ldots, E_k) \overset{a}{\longrightarrow} (F_1, \ldots, F_k)$.

For the only if case, we use two observations. The first observation is that whenever $(E_1, \ldots, E_k) \overset{a}{\longrightarrow} (F_1, \ldots, F_k)$ then any common implementation I of $E_1, \ldots, E_k$ has to have an $\overset{a}{\longrightarrow}$ transition to a common implementation J of $F_1, \ldots, F_k$. This is easily seen from the modal game characterization. As at least one $E_i \overset{a}{\longrightarrow} F_i$, by playing this transition Attacker enforces $I \overset{a}{\longrightarrow} J$. By playing $I \overset{a}{\dashrightarrow} J$ on the other side, Attacker then enforces J to be a common implementation of $F_1, \ldots, F_k$ as all these processes are deterministic and Defender has no other choice playing on their side. The second observation is that whenever $(G_1, \ldots, G_k)$ is a marked node, then there can be no common implementation of $G_1, \ldots, G_k$ which is clear from the definition of the graph. By considering these observations together, we can conclude that if a marked node is reachable from $(D_1, \ldots, D_k)$ then there can be no common implementation of $D_1, \ldots, D_k$. □

We have thus established an upper bound on the complexity of CI_D. As the space complexity is polynomial in k and logarithmic in n, we have proved Theorem 5.1.

We shall now prove that the upper bounds in this theorem are tight, i.e. that CI_D is PSPACE-complete and CI_D^k for any fixed k is NL-complete. The latter claim follows from the fact that deciding bisimilarity on finite deterministic processes is NL-complete (see the proof of Theorem 4.3) and an earlier observation that bisimilarity is a special case of common implementation for two processes, which are already implementations. Thus, we get the following result.

Theorem 5.3. *The problem* CI_D^k *for any fixed k is NL-hard.*

The remaining hardness result regarding the CI_D problem is asserted by the following theorem.

Theorem 5.4. *The problem* CI_D *is PSPACE-hard.*

The proof is done by reduction from the acceptance problem for deterministic linear bounded automata (LBA). A *deterministic LBA* is a tuple $\mathcal{M} = (Q, \Sigma, \Gamma, \triangleright, \triangleleft, q_0, q_{acc}, q_{rej}, \delta)$ where Q is a finite set of states, Σ is a finite input alphabet, $\Gamma \supseteq \Sigma$ is a finite tape alphabet, $\triangleright, \triangleleft \in \Gamma \setminus \Sigma$ are the left and right end markers, $q_0, q_{acc}, q_{rej} \in Q$ are the initial, accept and reject states, respectively, and $\delta : Q \setminus \{q_{acc}, q_{rej}\} \times \Gamma \to Q \times \Gamma \times \{L, R\}$ is a computation step function, such that whenever $\delta(q, X) = (q', Y, d)$ and one of X, Y is $\triangleright$ then both X and Y are $\triangleright$ and $d = R$; similarly if one of X, Y is $\triangleleft$ then both X and Y are $\triangleleft$ and $d = L$. We can w.l.o.g. assume that the input alphabet is binary, that is $\Sigma = \{a, b\}$ and that the tape alphabet only contains symbols from the input alphabet and the end markers, that is $\Gamma = \{a, b, \triangleright, \triangleleft\}$.

A configuration of $\mathcal{M}$ is given by the state, the position of the head and the content of the tape; we write it as a triple from $Q \times \mathbb{N} \times \Gamma^*$. A step of computation is a relation between configurations, denoted by $\vdash$, that is defined using the δ function in the usual way. Given a word $w \in \Sigma^*$, the initial configuration of $\mathcal{M}$ is $(q_0, 0, \triangleright w \triangleleft)$. A configuration is called accepting, if it is of the form (q_{acc}, i, z), and is called rejecting, if it is of the form (q_{rej}, i, z). A computation of $\mathcal{M}$ on a word w is a maximal sequence of configurations that begins with the initial configuration $(q_0, 0, \triangleright w \triangleleft)$ and such that the computational step relation holds between any two successive configurations. The machine $\mathcal{M}$ accepts a word $w \in \Sigma^*$, if the computation of $\mathcal{M}$ on w ends in an accepting configuration.

The computation of an LBA is unique and in what follows we assume that it is always finite (as any deterministic LBA with infinite computations can be transformed into an equivalent non-looping deterministic LBA). The problem whether a given deterministic LBA $\mathcal{M}$ accepts a given word $w \in \Sigma^*$ is PSPACE-complete (see e.g. [17]).

We can now proceed with the description of the reduction. Let $\mathcal{M}$ be a deterministic LBA and $w = w_1 w_2 \ldots w_n$ an input word of length n. We construct an $(n+3)$-tuple of deterministic processes $(D_{ctrl}, D_0, D_1, \ldots, D_n, D_{n+1})$ such that they have a common implementation if and only if $\mathcal{M}$ accepts w. Each of the D_i processes simulates one tape cell, the D_{ctrl} process simulates the control unit and the head. The action alphabet of the processes is $Act = \{a, b, r, \triangleright, \triangleleft, t_0, t_{n+1}, s^0_\triangleright, s^{n+1}_\triangleleft\} \cup \{t_i, s^i_a, s^i_b \mid 1 \leq i \leq n\}$.

For all i from 1 to n, the MTS underlying D_i has the set of processes $\{P^i_a, P^i_b, T^i_a, T^i_b\}$ and the transitions are defined as:

$$P^i_a \overset{t_i}{\dashrightarrow} T^i_a \qquad P^i_b \overset{t_i}{\dashrightarrow} T^i_b \qquad T^i_a \overset{a}{\dashrightarrow} P^i_a \qquad T^i_b \overset{b}{\dashrightarrow} P^i_b \qquad P^i_a \overset{x}{\dashrightarrow} P^i_a$$

$$P^i_a \overset{s^i_b}{\dashrightarrow} P^i_b \qquad P^i_b \overset{s^i_a}{\dashrightarrow} P^i_a \qquad T^i_a \overset{a}{\longrightarrow} P^i_a \qquad T^i_b \overset{b}{\longrightarrow} P^i_b \qquad P^i_b \overset{y}{\dashrightarrow} P^i_b$$

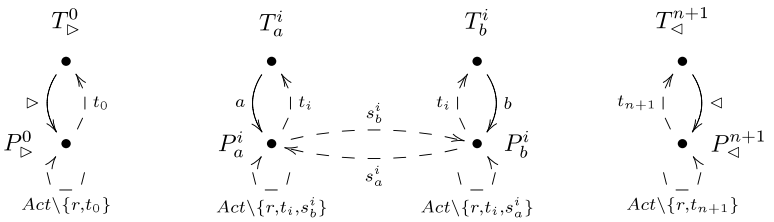

Fig. 10. MTS underlying D_0, D_i for $1 \leq i \leq n$, and D_{n+1}.

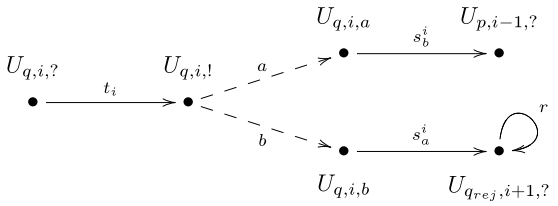

Fig. 11. An example of transitions for processes $U_{q,i,x}$.

for all $x \in Act \setminus \{r, t_i, s^i_b\}$ and $y \in Act \setminus \{r, t_i, s^i_a\}$. The process D_i is then defined as $D_i = P^i_{w_i}$. The processes D_0 and D_{n+1} are defined as $D_0 = P^0_{\triangleright}$ and $D_{n+1} = P^{n+1}_{\triangleleft}$ with transitions:

$$P^0_{\triangleright} \overset{x}{\dashrightarrow} P^0_{\triangleright} \quad T^0_{\triangleright} \overset{\triangleright}{\dashrightarrow} P^0_{\triangleright} \quad P^{n+1}_{\triangleleft} \overset{y}{\dashrightarrow} P^{n+1}_{\triangleleft} \quad T^{n+1}_{\triangleleft} \overset{\triangleleft}{\dashrightarrow} P^{n+1}_{\triangleleft}$$
$$P^0_{\triangleright} \overset{t_0}{\dashrightarrow} T^0_{\triangleright} \quad T^0_{\triangleright} \overset{\triangleright}{\longrightarrow} P^0_{\triangleright} \quad P^{n+1}_{\triangleleft} \overset{t_{n+1}}{\dashrightarrow} T^{n+1}_{\triangleleft} \quad T^{n+1}_{\triangleleft} \overset{\triangleleft}{\longrightarrow} P^{n+1}_{\triangleleft}$$

for all $x \in Act \setminus \{r, t_0\}$ and $y \in Act \setminus \{r, t_{n+1}\}$. The MTSs underlying D_i are shown in Fig. 10.

The MTS underlying D_{ctrl} is defined as follows. The set of processes is $\{U_{q,i,\alpha} \mid q \in Q, 0 \leq i \leq n+1, \alpha \in \{a, b, \triangleright, \triangleleft, ?, !\}\}$. The transitions are defined as:

$$U_{q,i,?} \overset{t_i}{\dashrightarrow} U_{q,i,!} \quad U_{q,i,!} \overset{z}{\dashrightarrow} U_{q,i,z} \quad U_{q,i,x} \overset{s^i_y}{\dashrightarrow} U_{p,j,?} \quad U_{q_{rej},i,?} \overset{r}{\dashrightarrow} U_{q_{rej},i,?}$$
$$U_{q,i,?} \overset{t_i}{\longrightarrow} U_{q,i,!} \quad U_{q,i,x} \overset{s^i_y}{\longrightarrow} U_{p,j,?} \quad U_{q_{rej},i,?} \overset{r}{\longrightarrow} U_{q_{rej},i,?}$$

for all $q \in Q \setminus \{q_{acc}, q_{rej}\}$, $0 \leq i \leq n+1$, for all $z \in \{a, b, \triangleright, \triangleleft\}$ and for all $x, y \in \{a, b, \triangleright, \triangleleft\}$ and $p \in Q$ such that $\delta(q, x) = (p, y, d)$ and $j = i - 1$ if $d = L$ and $j = i + 1$ if $d = R$. Other processes (most notably $U_{q_{acc},i,?}$) have no transitions. The process D_{ctrl} is then defined as $U_{q_0,0,?}$. This construction is illustrated by an example in Fig. 11. The depicted transitions represent the steps $\delta(q, a) = (p, b, L)$ and $\delta(q, b) = (q_{rej}, a, R)$.

What remains to be proved is the correctness of this construction. Before we do that, we prove a useful lemma about correspondence between configuration steps and $(n + 1)$-tuples of processes. This correspondence is represented by the following mapping φ (note that $z_0 = \triangleright$ and $z_{n+1} = \triangleleft$).

$$\varphi(q, i, z_0 \ldots z_{n+1}) = (U_{q,i,?}, P^0_{z_0}, \ldots, P^{n+1}_{z_{n+1}}).$$

Lemma 5.5. *Let (q, i, z) and (q', i', z') be two consecutive configurations and let I be a common implementation of $\varphi(q, i, z)$. Then any path going out from I has at least three transitions and moreover, after any three transitions the implementation I changes into a common implementation of $\varphi(q', i', z')$.*

Proof. Clearly, if (q, i, z) and (q', i', z') are two consecutive configurations then either $i' = i - 1$ (and $i \geq 1$) or $i' = i + 1$ (and $i \leq n$), and z and z' can only differ on their ith position. Moreover, this step is according to the function δ such that $\delta(q, z_i) = (q', z'_i, d)$ where either $d = R$ (if $i' = i + 1$) or $d = L$ (if $i' = i - 1$). The proof will use the game characterization of the modal refinement.

Let I be a common implementation of $(U_{q,i,?}, P^0_{z_0}, \ldots, P^{n+1}_{z_{n+1}})$. Attacker can force three steps of I by playing the t_i transition of $U_{q,i,?}$, then the z_i transition of $T^i_{z_i}$ and finally the $s^i_{z'_i}$ transition of U_{q,i,z_i}. Moreover, I can have no other behaviour than that starting with the sequence $t_i, z_i, s^i_{z'_i}$ and whenever I does these three steps, it changes into J where J is a common implementation of $(U_{q',i',?}, P^0_{z_0}, \ldots, P^{i-1}_{z_{i-1}}, P^i_{z'_i}, P^{i+1}_{z_{i+1}}, \ldots, P^{n+1}_{z_{n+1}})$, which is exactly $\varphi(q', i', z')$. Both these properties of I can be enforced by Attacker playing on the side of I. □

Lemma 5.6. *Let $\mathcal{M}$ be a deterministic LBA and $w = w_1 w_2 \ldots w_n$. Then $\mathcal{M}$ accepts w if and only if $(U_{q_0,0,?}, P^0_{\triangleright}, P^1_{w_1}, \ldots, P^n_{w_n}, P^{n+1}_{\triangleleft})$ have a common implementation.*

Proof. We first note that there is no common implementation of the processes $(U_{q_{rej},p,?}, P^0_{z_0}, \ldots, P^{n+1}_{z_{n+1}})$ as none of the $P^i_{z_i}$ processes allows the transition $\overset{r}{\dashrightarrow}$ whereas $U_{q_{rej},p,?}$ requires it. On the other hand there is always a common implementation of $(U_{q_{acc},p,?}, P^0_{z_0}, \ldots, P^{n+1}_{z_{n+1}})$—it is simply the implementation with no transitions at all. If $\mathcal{M}$ accepts w then the existence of a common implementation is a straightforward application of Lemma 5.5. For the other direction, if $\mathcal{M}$ rejects w, Lemma 5.5 shows that any common implementation must be able to reach a state that implements $(U_{q_{rej},i,?}, P^0_{z_0}, \ldots, P^{n+1}_{z_{n+1}})$, but there is no such common implementation. □

Corollary 5.7. *The problem* $\mathrm{CI}^k_{\mathrm{D}}$ *is NL-complete for any fixed* k *and* CI_{D} *is PSPACE-complete.*

6. Complexity results for deterministic implementation problem

In this section we investigate the problem whether a given collection of (nondeterministic) processes have a common deterministic implementation. This problem is computationally hard (EXPTIME-complete) not only for an arbitrary number of processes but also for a fixed number of them. In fact, we show that it is EXPTIME-complete even for single process and the question whether it has a deterministic implementation or not.

Definition 6.1. Let S be a (possibly nondeterministic) process. The set of *deterministic implementations* of S, denoted by $[\![S]\!]_{\mathrm{D}}$, is defined as $[\![S]\!]_{\mathrm{D}} = [\![S]\!] \cap d\mathcal{MTS}$.

The problems that we study in this section are defined as follows.

$$\mathrm{dCI}^k = \{\langle S_1, \ldots, S_k\rangle \mid \exists I : I \in [\![S_1]\!]_{\mathrm{D}} \cap \cdots \cap [\![S_k]\!]_{\mathrm{D}}\}$$
$$\mathrm{dCI} = \bigcup_{k=1}^{\infty} \mathrm{dCI}^k.$$

We shall now argue that dCI is EXPTIME-complete and later on use this fact to conclude that dCI^k is also EXPTIME-complete, even for $k = 1$.

In order to capture what a common deterministic implementation of a given set of (nondeterministic) processes has to fulfill, we introduce the following notion of a *possible successor*. Consider a deterministic I implementing all processes from a set $\mathcal{S}$. Then some must a-successor of I has to implement all must a-successors of the processes from $\mathcal{S}$, and moreover, when some of the processes in $\mathcal{S}$ do not have any must a-successors then they need to have at least a may a-successor in order to match the must transition of the implementation. We formalize this consideration by the following definition of the set of all possible a-successors of $\mathcal{S}$.

$$\mathrm{MustSucc}_a(\mathcal{S}) = \{S' \mid \exists S \in \mathcal{S} : S \xrightarrow{a} S'\}$$
$$\mathrm{PossibleSucc}_a(\mathcal{S}) = \{\mathrm{MustSucc}_a(\mathcal{S}) \cup \mathcal{T} \mid \forall S \in \mathcal{S}.\ \exists T \in \mathcal{T} : S \overset{a}{\dashrightarrow} T\}.$$

Definition 6.2. A *set of deterministically consistent subsets* (DCS) on a set of processes P of an MTS $M = (P, \dashrightarrow, \longrightarrow)$ is a set $R \subseteq \mathcal{P}(P)$ such that for every action a, whenever $\mathcal{S} \in R$ and $\mathrm{MustSucc}_a(\mathcal{S}) \neq \emptyset$ then $\mathrm{PossibleSucc}_a(\mathcal{S}) \cap R \neq \emptyset$.

In other words, if every common implementation of processes in $\mathcal{S}$ has to continue (one of the processes has a must transition) then there has to be a possible successor in R, which thus again has a deterministic continuation.

Since the union of DCSs is again a DCS, we can consider the greatest DCS.

Definition 6.3. Let M be an MTS. By R_M we denote the greatest set of deterministically consistent subsets of P.

Lemma 6.4. *Let* M *be an MTS, then* R_M *contains precisely those sets of its processes that have a common deterministic implementation. Moreover,* R_M *is computable in EXPTIME.*

Proof. *Soundness.* Let $\mathcal{S} \in R_M$. We construct a deterministic common implementation of all processes in $\mathcal{S}$. Let $M_d = (R_M, \longrightarrow, \longrightarrow)$ be an MTS where the transitions are given as follows:

> for every action a and $\mathcal{T} \in R_M$, if $\mathrm{MustSucc}_a(\mathcal{T}) \neq \emptyset$ then we set $\mathcal{T} \xrightarrow{a} \mathcal{T}'$ for some arbitrary (but fixed) $\mathcal{T}' \in \mathrm{PossibleSucc}_a(\mathcal{T})$.

This is a deterministic refinement of M with the refining relation $\{(\mathcal{T}, T) \mid \mathcal{T} \in R_M, T \in \mathcal{T}\}$, since a transition in the implementation is always allowed by all processes in $\mathcal{T}$, in particular by T, and the implementation includes all must-successors of T, too. Hence $\mathcal{S}$, as a process of M_d, is the desired common deterministic implementation of all processes from $\mathcal{S}$.

Completeness. Let $\mathcal{S}$ be a set of processes having a common deterministic implementation I. Assume that each process J reachable from I is labelled by the set of all processes of M that J implements. We show that the set R, consisting of all labels of processes reachable from I, is a DCS on M. For technical convenience, we identify the names of the processes reachable from I with their labels.

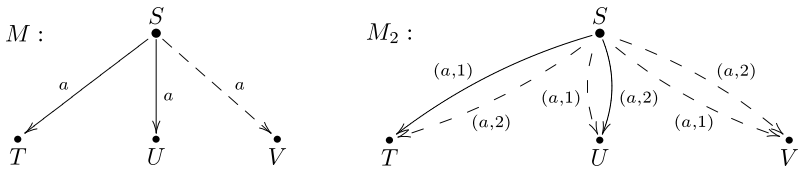

Fig. 12. Input instance M of CI and the constructed instance M_B of dCI where $B = 2$.

Since every $\mathcal{T} \in R$ has a deterministic implementation, then for each action a, if $\mathrm{MustSucc}_a(\mathcal{T}) \neq \emptyset$ then there is precisely one a-transition $\mathcal{T} \xrightarrow{a} \mathcal{T}'$ for some $\mathcal{T}'$. Because $\mathcal{T}$ is a common implementation of all processes from $\mathcal{T}$, we have $\mathrm{MustSucc}_a(\mathcal{T}) \subseteq \mathcal{T}'$ and for every $T \in \mathcal{T}$ there is $T \xrightarrow{a} T'$ with $T' \in \mathcal{T}'$. We can so conclude that $\mathcal{T}' \in R$.

Complexity. We can compute R_M in exponential time by the standard co-inductive algorithm: we begin with including all sets of processes and then we keep repeatedly removing any inconsistent sets, until we reach a fixed point, giving us exactly R_M. The exponential running time follows from the fact that $|\mathcal{P}(P)| = 2^{|P|}$. □

We now turn our attention to the hardness of the deterministic common implementation problem and provide a reduction for the EXPTIME-complete problem CI (see [6]) to dCI. We have to modify the given instance of common implementation problem such that the instance has a (nondeterministic) common implementation if and only if the newly constructed instance of dCI has a deterministic common implementation. We proceed in two steps. First, we modify the given processes (instance of CI) so that their new must transition relation does not include more than one transition under the same action while preserving the (non)existence of a common implementation. Second, we prove that CI and dCI coincide on MTSs with such must transition relation.

We start with the description of the modification of the input processes for the CI problem. Let $M = (P, \dashrightarrow, \longrightarrow)$ be their underlying MTS over an alphabet Σ and let B be the size of the $\longrightarrow$ relation. We assign different numbers from 1 to B to the must transitions and denote this assignment function by f. We now construct an MTS $M_B = (P, \dashrightarrow_B, \longrightarrow_B)$ over the alphabet $\overline{\Sigma} = \Sigma \times \{1, \ldots, B\}$. The new must transitions are now distinguished by their indices, and all may transitions are now allowed under all possible indices. Formally,

- for every $S \xrightarrow{a} T$ we set $S \xrightarrow{(a, f(S \xrightarrow{a} T))}_B T$, and
- for every $S \dashrightarrow^{a} T$ we set $S \dashrightarrow^{(a,i)}_B T$ for all $1 \leq i \leq B$.

Note that after the transformation if $S_1 \xrightarrow{(a,i)} T_1$ and $S_2 \xrightarrow{(a,i)} T_2$ then $S_1 = S_2$ and $T_1 = T_2$. An example of the reduction is given in Fig. 12.

Lemma 6.5. *Processes $S_1, \ldots, S_k \in P$ have a common implementation as processes of M iff they have a common implementation as processes of M_B.*

Proof. For the only if part, let I be a common implementation of $S_1, \ldots, S_k$ as processes of M. We change the labelling of the transitions so that it becomes an implementation I_B of $S_1, \ldots, S_k \in P$ as processes of M_B. If there is an a-transition in I, we put there (a, i)-transitions for all indices $i \in \{1, \ldots, B\}$. Now I_B is a common implementation of $S_1, \ldots, S_k$ in M_B since all must transitions are implemented, and the may transitions in the implementation I_B originated from I, thus being implementations of the original may transitions, which are now allowed as a pair with all possible indices in the second component.

For the if part, as M is equivalent to M_B where indices are forgotten, the common implementation of processes of M_B is turned into a common implementation of processes of M simply by forgetting the indices too. □

In the following, we show that if there are no must transitions under the same action, then we can modify any common (nondeterministic) implementation into a deterministic one.

Lemma 6.6. *Let $M = (P, \dashrightarrow, \longrightarrow)$ be an MTS such that for every processes $S_1, S_2, T_1, T_2 \in P$ and any action a, if $S_1 \xrightarrow{a} T_1$ and $S_2 \xrightarrow{a} T_2$ then $S_1 = S_2$ and $T_1 = T_2$. Then, for every $S_1, \ldots, S_k \in P$, if $[\![S_1]\!] \cap \cdots \cap [\![S_k]\!] \neq \emptyset$ then $[\![S_1]\!]_D \cap \cdots \cap [\![S_k]\!]_D \neq \emptyset$.*

Proof. Let I be a common implementation with R being a refinement relation containing (I, S_i) for $1 \leq i \leq k$. We show that we can prune I so that we get a deterministic common implementation. For every process J from the underlying system of I and an action a, if there are (unique) $T, T' \in P$ with $T \xrightarrow{a} T'$ and $(J, T) \in R$, then there is at least one $J \xrightarrow{a} J'$ with $(J', T') \in R$, we keep this a-transition in J and omit the others; otherwise, we omit all a-transitions from J.

We show that R is still a refinement relation (containing (I, S_i) for $1 \leq i \leq k$). Since the new may transition relation in I is smaller, we only need to show that all must transitions are still realized. Let $(J, T) \in R$ and $T \xrightarrow{a} T'$. Such T and T' are unique, hence the respective transition $J \xrightarrow{a} J'$ with (J', T') has been preserved. □

Corollary 6.7. *The problem* dCI *is EXPTIME-complete.*

$$\frac{I \xrightarrow{a} I' \quad J \xrightarrow{a} J'}{I \parallel J \xrightarrow{a} I' \parallel J'} \quad \forall a \in \Sigma$$

a

$$\frac{S \xrightarrow{a} S' \quad T \xrightarrow{a} T'}{S \parallel T \xrightarrow{a} S' \parallel T'} \quad \forall a \in \Sigma$$

$$\frac{S \dashrightarrow^{a} S' \quad T \dashrightarrow^{a} T'}{S \parallel T \dashrightarrow^{a} S' \parallel T'} \quad \forall a \in \Sigma$$

b

Fig. 13. (a) Simple parallel operator (b) Its lift to $\mathcal{MTS}$.

Proof. The containment follows from Lemma 6.4, and the hardness from Lemmas 6.5 and 6.6 and the fact that M_B has must transition relation with every transition having a unique label. □

We are now ready to prove the equivalence of dCI and dCI^1 and conclude with the following theorem.

Theorem 6.8. *The problem* dCI^1 *is EXPTIME-complete.*

Proof. We show that dCI reduces to dCI^1. Consider $S_1, \ldots, S_k$. We construct a new process S such that

$$S \xrightarrow{a} S_1,\ S \xrightarrow{a} S_2,\ \ldots,\ S \xrightarrow{a} S_k$$

for some action a. Clearly, $S_1, \ldots, S_k$ have a common deterministic implementation iff S has a deterministic implementation. □

7. Composition operators

We shall now discuss composition operators on labelled transition systems and extend them in order to apply them on modal transition systems. We recall that implementations are labelled transition systems where two identical copies of the transition relation are considered and we shall not distinguish explicitly between the two copies of the transition relation. Therefore, as this operator extension should treat implementations in a similar way as labelled transition systems, the operators should transform may and must relations in the same way. Moreover, when applied on a general MTS, the operators have to preserve the inclusion of must relation in the may relation in order to obtain a correct MTS as a result. This is guaranteed by monotonicity of operators.

After defining the extension of operators, we discuss the relation between this extension on MTSs and the direct application on the respective sets of implementations.

Definition 7.1. An n-ary *operator on* $i\mathcal{MTS}$ is a class mapping $i\mathcal{MTS}^n \to i\mathcal{MTS}$. An n-ary operator $\oplus$ on $i\mathcal{MTS}$ is *liftable* if

1. the resulting process set does not depend on the input transition relation (it is usually e.g. the Cartesian product or the sum), and
2. it is *covariant monotonous*, i.e. for all implementations $I_i = (P_i, \longrightarrow_i)$ and $J_i = (P_i, \hookrightarrow_i)$ such that $1 \leq i \leq n$ and where we let $\oplus_i I_i = (P, \longrightarrow)$ and $\oplus_i J_i = (P, \hookrightarrow)$, whenever $\longrightarrow_i \subseteq \hookrightarrow_i$ for all i, $1 \leq i \leq n$, then also $\longrightarrow \subseteq \hookrightarrow$.

An n-ary *operator on* $\mathcal{MTS}$ is a class mapping $\mathcal{MTS}^n \to \mathcal{MTS}$.

A *syntactic lift* of an n-ary liftable operator $\oplus$ on $i\mathcal{MTS}$ is an n-ary operator $\oplus^M$ on $\mathcal{MTS}$ defined as follows. Let $(P_i, \dashrightarrow_i, \longrightarrow_i)$ be MTS for $i \in \{1, \ldots, n\}$ and let $\oplus_{1\leq i\leq n}(P_i, \dashrightarrow_i) = (P, \dashrightarrow)$ and $\oplus_{1\leq i\leq n}(P_i, \longrightarrow_i) = (P, \longrightarrow)$. We define $\oplus^M_{1\leq i\leq n}(P_i, \dashrightarrow_i, \longrightarrow_i) = (P, \dashrightarrow, \longrightarrow)$. (The resulting system is an MTS due to the covariant monotonicity.)

An n-ary *operator on sets of implementations* is a class mapping that maps n-tuples of classes of implementations to a class of implementations.

A *semantic lift* of an n-ary operator $\oplus$ on $i\mathcal{MTS}$ is an n-ary operator $\oplus^I$ on sets of implementations defined by $\oplus^I_i \mathcal{I}_i = \{\oplus_i I_i \mid I_i \in \mathcal{I}_i\}$.

We often omit the letters M and I when it is clear which lift is meant. We give an example of a parallel operator and its lift in Fig. 13. This is an important example, as it is quite simple while capturing the basic aspects of the concept of a parallel composition of processes.

Note that if we added rules such as

$$\text{if } A \xrightarrow{c} A' \text{ and } B \not\xrightarrow{c} \text{ then } A \parallel B \xrightarrow{c} A' \parallel B$$

then the operator would cease to be monotonous and hence liftable. It would indeed transform MTSs into incorrect systems.

We want the liftable operators to maintain the so-called independent implementability, i.e. whenever $I_i \in [\![S_i]\!]$ then also $\oplus_i I_i \in [\![\oplus_i S_i]\!]$. To ensure this, we only require the operators to behave compositionally. The most natural way to guarantee this is to define the operators syntactically by a set of rules.

$$\frac{I \xrightarrow{a} I'}{I + J \xrightarrow{a} I'} \quad \forall a \in \Sigma \qquad \frac{J \xrightarrow{a} J'}{I + J \xrightarrow{a} J'} \quad \forall a \in \Sigma$$

a

$$\frac{S \xrightarrow{a} S'}{S + T \xrightarrow{a} S'} \qquad \frac{S \xrightarrow{a} S'}{S + T \xrightarrow{a} S'} \quad \forall a \in \Sigma$$

$$\frac{T \xrightarrow{a} T'}{S + T \xrightarrow{a} T'} \qquad \frac{T \xrightarrow{a} T'}{S + T \xrightarrow{a} T'} \quad \forall a \in \Sigma$$

b

Fig. 14. (a) Nondeterministic sum (b) Its lift to $\mathcal{MTS}$.

Definition 7.2. Let Σ be a set of actions such that $\bullet \notin \Sigma$ and $n \in \mathbb{N}$. An n-ary *context system* over Σ is a tuple $\mathcal{C} = (C, c_0, \rho)$ where C is a nonempty set of *contexts*, $c_0 \in C$ is an *initial context* and $\rho \subseteq C \times \Sigma \times (\Sigma \cup \{\bullet\})^n \times C$ is a set of *rules*.

Given an n-ary context system $\mathcal{C}$, we can define a composition of n $i\mathcal{MTS}$s, denoted as $|_{\mathcal{C}}\ (P_i, \longrightarrow_i)_{i\in\{1,\ldots,n\}}$ in the following way:

$$|_{\mathcal{C}}\ (P_i, \longrightarrow_i)_{i\in\{1,\ldots,n\}} = \left(C \times \prod_{i=1}^{n} P_i, \longrightarrow\right)$$

where $(c, p_1, \ldots, p_n) \xrightarrow{a} (c', p'_1, \ldots, p'_n)$ whenever $(c, a, (a_1, \ldots, a_n), c') \in \rho$ and for all i, $p_i \xrightarrow{a_i} p'_i$ where we assume the convention that $p \xrightarrow{\bullet} p'$ if and only if p and p' are identical. The composition of n processes $I_1, \ldots, I_n$ is then defined to be the process $|_{\mathcal{C}}\ (I_i)_{i\in\{1,\ldots,n\}} = (c_0, I_1, \ldots, I_n)$. The operator $|_{\mathcal{C}}$ is called the *general composition operator* with context system $\mathcal{C}$.

Example 7.3. Clearly, the $\|$ operator defined in Fig. 13 can be seen as a general composition operator. Indeed, if we take $\mathcal{C} = (\{c\}, c, \{(c, a, (a, a), c) \mid a \in \Sigma\})$ then $\|$ is identical with $|_{\mathcal{C}}$.

To show an example of a general composition operator requiring more contexts, let us take the usual definition of a nondeterministic sum of processes, depicted in Fig. 14. This operator can be defined as a general composition operator with context (C, c_0, ρ) where $C = \{+, 1, 2\}, c_0 = +$ and $\rho = \{(+, a, (a, \bullet), 1), (+, a, (\bullet, a), 2), (1, a, (a, \bullet), 1), (2, a, (\bullet, a), 2) \mid a \in \Sigma\}$.

Remark 7.4. The general composition operators are clearly liftable.

These operators guarantee independent implementability. To simplify the technical arguments, the following lemma proves that for pairs and on one side only. However, the general proof is a straightforward extension.

Lemma 7.5. *The modal refinement relation is a precongruence for any general composition operator, that is if* $S_1 \leq_m S_2$ *then* $S_1 |_{\mathcal{C}} T \leq_m S_2 |_{\mathcal{C}} T$ *for any process* T *and any context system* $\mathcal{C}$.

Proof. Let $\mathcal{C} = (C, c_0, \rho)$ be a context system, $|_{\mathcal{C}}$ the general composition operator with context $\mathcal{C}$, T an arbitrary process and S_1, S_2 processes such that $S_1 \leq_m S_2$. We will show a relation R that satisfies Definition 2.1 such that $(S_1 |_{\mathcal{C}} T, S_2 |_{\mathcal{C}} T) \in R$.

We define $R = \{((c, U_1, V), (c, U_2, V)) \mid c \in C, U_1 \leq_m U_2', V$ arbitrary process in MTS underlying T$\}$ and we show that the conditions of refinement relation are satisfied.

(i) Clearly $(S_1 |_{\mathcal{C}} T, S_2 |_{\mathcal{C}} T) = ((c_0, S_1, T), (c_0, S_2, T)) \in R$.

(ii) Suppose that $(c, U_1, V) \overset{a}{\dashrightarrow} (c', U_1', V')$. Then $(c, a, (\alpha, \beta), c') \in \rho, U_1 \overset{\alpha}{\dashrightarrow} U_1'$ and $V \overset{\beta}{\dashrightarrow} V'$. As $U_1 \leq_m U_2$ this means that $U_2 \overset{\alpha}{\dashrightarrow} U_2'$ and $U_1' \leq_m U_2'$. Then also $(c, U_2, V) \overset{a}{\dashrightarrow} (c', U_2', V')$ and $((c', U_1', V'), (c', U_2', V')) \in R$.

(iii) Suppose that $(c, U_2, V) \xrightarrow{a} (c', U_2', V')$. Then $(c, a, (\alpha, \beta), c') \in \rho, U_2 \xrightarrow{\alpha} U_2'$ and $V \xrightarrow{\beta} V'$. As $U_1 \leq_m U_2$ this means that $U_1 \xrightarrow{\alpha} U_1'$ and $U_1' \leq_m U_2'$. Then also $(c, U_1, V) \xrightarrow{a} (c', U_1', V')$ and $((c', U_1', V'), (c', U_2', V')) \in R$. □

Corollary 7.6. *For all processes* S_1 *and* S_2 *we have* $[\![S_1]\!] |_{\mathcal{C}} [\![S_2]\!] \subseteq [\![S_1 |_{\mathcal{C}} S_2]\!]$.

Proof. Let $I_1 \leq_m S_1$ and $I_2 \leq_m S_2$. Then $I_1 |_{\mathcal{C}} I_2 \leq_m I_1 |_{\mathcal{C}} S_2 \leq_m S_1 |_{\mathcal{C}} S_2$. □

Remark 7.7. A related general approach of defining composition operators has been studied in [19]. It can be shown that any operator defined this way can be written as a general composition operator with (possibly infinite) context, as clearly we can take the set of all terms with n free variables as the set of contexts. Thus, the above results hold for any operators defined using [19].

Example 7.8. As has been mentioned in Example 7.3, $\|$ is a general composition operator, hence

$$[\![S_1]\!] \,\|\, [\![S_2]\!] \subseteq [\![S_1 \,\|\, S_2]\!]. \qquad (*)$$

However, the inclusion may be strict. In Fig. 15 we can see an example of two *deterministic* processes S_1 and S_2 such that $[\![S_1]\!] \,\|\, [\![S_2]\!] \subsetneq [\![S_1 \,\|\, S_2]\!]$. Indeed, there is no implementation from $[\![S_1]\!] \,\|\, [\![S_2]\!]$ that would be even bisimilar to I.

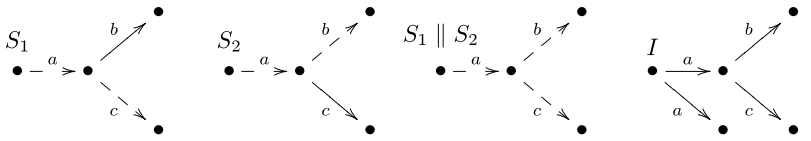

Fig. 15. $I \in [\![S_1 \parallel S_2]\!]$ but $I \notin [\![S_1]\!] \parallel [\![S_2]\!]$.

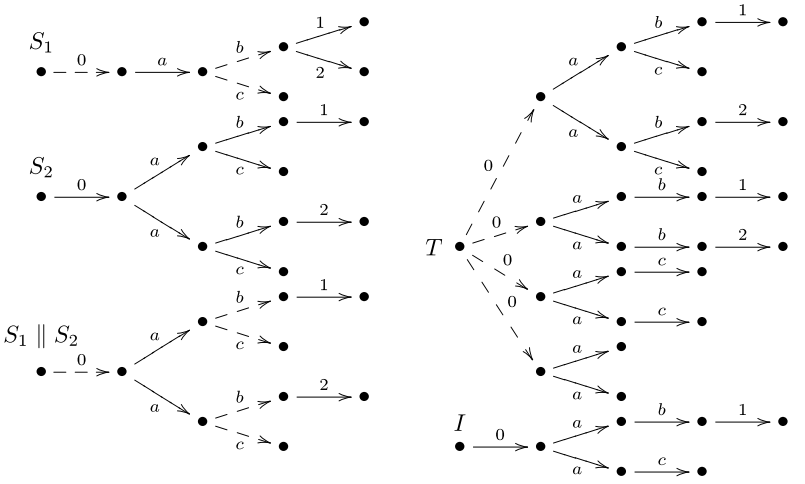

Fig. 16. $[\![S_1]\!] \parallel [\![S_2]\!] \subseteq [\![T]\!]$ and $I \in [\![S_1 \parallel S_2]\!] \setminus [\![T]\!]$.

In fact, there is no MTS describing the semantic composition $[\![S_1]\!] \parallel [\![S_2]\!]$ in this case. Observe that a sum of implementations of a system is again an implementation of this system. We thus conclude that I (being a sum of two implementations from $[\![S_1]\!] \parallel [\![S_2]\!]$) is necessarily an implementation of any system implementing all elements of $[\![S_1]\!] \parallel [\![S_2]\!]$.

However, there may exist better approximations of the semantic composition than the syntactic composition, particularly those describing consistencies of branches up to a finite number of steps. In Fig. 16, we show processes S_1, S_2, T such that $[\![S_1]\!] \parallel [\![S_2]\!] \subseteq [\![T]\!] \subsetneq [\![S_1 \parallel S_2]\!]$.

As there can be a strict inclusion between the semantic and syntactic composition in $(*)$ even in the case when specifications are deterministic, we investigate the case when only deterministic implementations are considered. We recall the notation of Definition 6.1: we write $I \in [\![S]\!]_D$ to denote that I is a deterministic implementation of (generally nondeterministic) S.

Lemma 7.9. *Let $M = (P, \dashrightarrow, \longrightarrow)$ be an MTS such that for every process $S \in P$ and every action a it holds that $S \overset{a}{\dashrightarrow} T$ and $S \overset{a}{\longrightarrow} U$ implies $[\![T]\!]_D \subseteq [\![U]\!]_D$. Then for every $S_1, S_2 \in P$ we get*

$$[\![S_1]\!]_D \parallel [\![S_2]\!]_D = [\![S_1 \parallel S_2]\!]_D.$$

Proof. We prove by coinduction that for all $I \in [\![S_1 \parallel S_2]\!]_D$ exist $I_1 \in [\![S_1]\!]_D$ and $I_2 \in [\![S_2]\!]_D$ such that $I = I_1 \parallel I_2$ (in this proof '$=$' is understood as an isomorphism on implementations). To construct I_1 and I_2 we perform the following for each action a.

- If $I \overset{a}{\longrightarrow} J$, then $S_1 \parallel S_2 \overset{a}{\dashrightarrow} T_1 \parallel T_2$ with $J \in [\![T_1 \parallel T_2]\!]_D$. By the coinductive hypothesis, there are $J_1 \in [\![T_1]\!]_D, J_2 \in [\![T_2]\!]_D$ such that $J = J_1 \parallel J_2$. We set $I_1 \overset{a}{\longrightarrow} J_1$ and $I_2 \overset{a}{\longrightarrow} J_2$.
- If $I \overset{a}{\not\longrightarrow}$, then also $S_1 \parallel S_2 \overset{a}{\not\longrightarrow}$ which means that at least one of $S_1 \overset{a}{\not\longrightarrow}, S_2 \overset{a}{\not\longrightarrow}$ has to hold. If both hold, we set $I_1 \overset{a}{\not\longrightarrow}$ and $I_2 \overset{a}{\not\longrightarrow}$, if only $S_1 \overset{a}{\not\longrightarrow}$ holds, we set $I_1 \overset{a}{\not\longrightarrow}$ and $I_2 \overset{a}{\longrightarrow} J$ where J is an arbitrary deterministic implementation of some T_2 such that $S_2 \overset{a}{\longrightarrow} T_2$, and similarly in the symmetric case.

Now clearly $I = I_1 \parallel I_2$, but it remains to prove that $I_1 \in [\![S_1]\!]_D$ and $I_2 \in [\![S_2]\!]_D$. We prove the first proposition, the other is symmetric.

(i) If $I_1 \overset{a}{\longrightarrow} J_1$ then clearly from the construction, there is some $S_1 \overset{a}{\dashrightarrow} T_1$ such that $J_1 \in [\![T_1]\!]_D$.

(ii) If $S_1 \overset{a}{\longrightarrow} U$ then clearly $I_1 \overset{a}{\longrightarrow} J_1$. We need to show that $J_1 \in [\![U]\!]_D$. But we know that $J_1 \in [\![T_1]\!]_D$ for some T_1 such that $S_1 \overset{a}{\dashrightarrow} T_1$. Using the premise of the lemma, we know that $[\![T_1]\!]_D \subseteq [\![U]\!]_D$, thus $J \in [\![U]\!]_D$. □

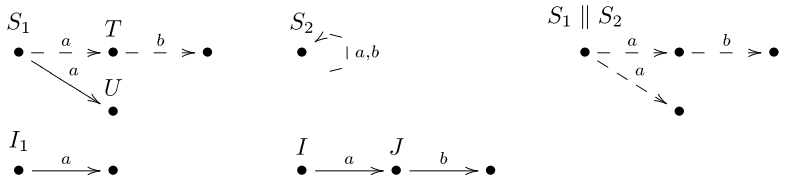

Fig. 17. $I \in [\![S_1 \parallel S_2]\!]_D$ but $I \notin [\![S_1]\!]_D \parallel [\![S_2]\!]_D$, because S_1 has only one implementation (up to bisimulation), and that is I_1.

The lemma requires each process to fulfill that all deterministic implementations of its successors are also implementations of its every must-successor. This conditions is, however, not a syntactic one and cannot be effectively checked. We are thus interested in possibly weaker, but syntactic condition. Since it is useless to have weaker mays than musts, we choose to require every process either not to have a must transition (several mays are possible) or to be deterministic (only one transition is possible and it can be must). This condition is clearly stronger than the premise of Lemma 7.9, the following theorem is therefore a corollary of the lemma.

Theorem 7.10. *Let $M = (P, \dashrightarrow, \longrightarrow)$ be an MTS such that for every process $S \in P$ and every action a holds: either $S \overset{a}{\not\longrightarrow}$, or $S \overset{a}{\dashrightarrow} T$ and $S \overset{a}{\dashrightarrow} U$ imply $T = U$. Then for every $S_1, S_2 \in P$ we get*

$$[\![S_1]\!]_D \parallel [\![S_2]\!]_D = [\![S_1 \parallel S_2]\!]_D.$$

If the criterion formulated in the above discussion does not hold, the equality is not guaranteed to hold either. Indeed, let $S_1 \overset{a}{\dashrightarrow} T$ and $S_1 \overset{a}{\longrightarrow} U$ with $[\![T]\!]_D \not\subseteq [\![U]\!]_D$ witnessed by $J \in [\![T]\!]_D \setminus [\![U]\!]_D$. If we take e.g. $S_2 \overset{b}{\dashrightarrow} S_2$ for all b, then $I \overset{a}{\longrightarrow} J$ is the counterexample, see Fig. 17.

8. Conclusion

We have studied several problems related to modal transition systems with a particular focus on the situation when the involved processes are deterministic. Apart from some fundamental results regarding the relationship between thorough and modal refinement, construction of the deterministic hull and a detailed discussion of composition operators liftable to the setting of modal transition systems, we contributed with the characterization of the computational complexity of several decision problems usually studied in the context of modal transition systems. In the following table we give an overview of the results related to deciding modal and thorough refinements for different combination of processes on the left- and right-hand side (here D stands for deterministic processes and N for nondeterministic processes). Complexity bounds proved in the present article are in bold.

	MR	TR
D,D	∈ **NL** **NL-hard**	∈ **NL** **NL-hard**
N,D	∈ **NL** **NL-hard**	∈ **NL** **NL-hard**
D,N	∈ P [12,13] **P-hard**	∈ EXPTIME [5] co-NP-hard [7]
N,N	∈ P [12,13] P-hard [14]	∈ EXPTIME [5] EXPTIME-hard [4]

We have also investigated the complexity of common implementation problems. Compared to the previously studied problem CI for arbitrary processes, we showed that the complexity improves when the given processes are deterministic (CI_D problem). Finally, the problems dCI and dCI^k, asking whether an arbitrary resp. a fixed number of nondeterministic processes have a common deterministic implementation, were proved to be EXPTIME-complete. In fact, this problem remains EXPTIME-complete even for a single nondeterministic process (asking whether it has at least one deterministic implementation or not). The following table provides the summary and, as before, our results (matching lower and upper bounds) are in bold.

	fixed number	arbitrary number
CI	P-complete [14,18]	EXPTIME-complete [6]
CI_D	**NL-complete**	**PSPACE-complete**
dCI	**EXPTIME-complete**	**EXPTIME-complete**

The results indicate that the complexity of several problems connected to the thorough refinement relation (which is more desirable in the refinement process than the modal refinement relation) become more tractable if the given specifications are deterministic (a standard assumption in much of the recent work, see e.g. [8,9]). On the other hand, the complexity in most instances does not improve if we consider deterministic implementations of nondeterministic specifications as we already mentioned that, for example, the question whether a given nondeterministic specification has at least one deterministic implementation is already EXPTIME-hard.

Acknowledgements

N. Beneš has been partially supported by the Grant Agency of the Czech Republic, grant No. GA201/09/1389. J. Křetínský has been partially supported by the research centre *Institute for Theoretical Computer Science (ITI)*, project No. 1M0545. J. Srba has been partially supported by Ministry of Education of the Czech Republic, project No. MSM 0021622419. K.G. Larsen has been partially supported by the VKR Center of Excellence MT-LAB.

References

[1] A. Antonik, M. Huth, K.G. Larsen, U. Nyman, A. Wasowski, 20 years of modal and mixed specifications, Bulletin of the EATCS 95 (2008) 94–129.
[2] K.G. Larsen, B. Thomsen, A modal process logic, in: LICS, IEEE Computer Society, 1988, pp. 203–210.
[3] H. Hüttel, K.G. Larsen, The use of static constructs in a modal process logic, in: A.R. Meyer, M.A. Taitslin (Eds.), Logic at Botik, in: Lecture Notes in Computer Science, vol. 363, Springer, 1989, pp. 163–180.
[4] N. Beneš, J. Křetínský, K. Larsen, J. Srba, Checking thorough refinement on modal transition systems is EXPTIME-complete, in: Proceedings of the 6th International Colloquium on Theoretical Aspects of Computing, LNCS, Springer-Verlag, 2009 (in press).
[5] A. Antonik, M. Huth, K.G. Larsen, U. Nyman, A. Wasowski, Complexity of decision problems for mixed and modal specifications, in: Proceedings of the 11th International Conference on Foundations of Software Science and Computation Structures, FOSSACS'08, in: LNCS, vol. 4962, 2008, pp. 112–126.
[6] A. Antonik, M. Huth, K.G. Larsen, U. Nyman, A. Wasowski, EXPTIME-complete decision problems for mixed and modal specifications, in: 15th International Workshop on Expressiveness in Concurrency, 2008.
[7] K.G. Larsen, U. Nyman, A. Wasowski, On modal refinement and consistency, in: Proceedings of the 18th International Conference on Concurrency Theory, CONCUR'07, in: LNCS, vol. 4703, Springer, 2007, pp. 105–119.
[8] T. Henzinger, J. Sifakis, The embedded systems design challenge, in: Proceedings of the 14th International Symposium on Formal Methods, FM'06, in: LNCS, vol. 4085, Springer-Verlag, 2006, pp. 1–15.
[9] T.A. Henzinger, J. Sifakis, The discipline of embedded systems design, IEEE Computer 40 (10) (2007) 32–40.
[10] C. Stirling, Local model checking games, in: Proceedings of the 6th International Conference on Concurrency Theory, CONCUR'95, in: LNCS, vol. 962, Springer-Verlag, 1995, pp. 1–11.
[11] W. Thomas, On the Ehrenfeucht-Fraïssé game in theoretical computer science (extended abstract), in: Proceedings of the 4th International Joint Conference CAAP/FASE, Theory and Practice of Software Development, TAPSOFT'93, in: LNCS, vol. 668, Springer-Verlag, 1993, pp. 559–568.
[12] P. Kanellakis, S. Smolka, CCS expressions, finite state processes, and three problems of equivalence, Information and Computation 86 (1) (1990) 43–68.
[13] R. Paige, R. Tarjan, Three partition refinement algorithms, SIAM Journal of Computing 16 (6) (1987) 973–989.
[14] J.L. Balcazar, J. Gabarró, M. Santha, Deciding bisimilarity is *P*-complete, Formal Aspects of Computing 4 (6 A) (1992) 638–648.
[15] Z. Sawa, P. Jančar, Behavioural equivalences on finite-state systems are PTIME-hard, Computing and Informatics 24 (5) (2005) 513–528.
[16] J.E. Hopcroft, R. Motwani, J.D. Ullman, Introduction to Automata Theory, Languages and Computability, Addison-Wesley Longman Publishing Co., Inc, Boston, MA, USA, 2000.
[17] C.H. Papadimitriou, Computational Complexity, Addison-Wesley Publishing Co., Inc., Reading, MA, USA, 1994.
[18] A. Hussain, M. Huth, On model checking multiple hybrid views, Tech. rep., department of Computer Science, University of Cyprus, TR-2004-6, 2004.
[19] R. De Simone, M. Nivat, Higher-level synchronising devices in MEIJE-SCCS, Theoretical Computer Science 37 (3) (1985) 245–267.